CAPÍTULO 5

PROCESSO

Prezado(a) aluno(a),

Este capítulo apresenta vários conceitos relacionados a processo. Entre eles estão a estrutura, os estados, o bloco de controle e as transições de estado do processo, além do relacionamento entre processos, os tipos de processos e o conceito de sinais.

Dê atenção especial a este capítulo, pois todos os conceitos apresentados aqui são importantes e serão utilizados nos capítulos seguintes. Além disso, é imprescindível que você entenda quais eventos causam as transições entre os estados do processo.

Bom estudo!

Fala Professor

5.1. Introdução

O conceito de processo é base para a implementação da multiprogramação, pois o processador é projetado apenas para executar instruções, não sendo capaz de distinguir qual programa se encontra em execução. A gerência de um ambiente multiprogramável é uma função exclusiva do sistema operacional, que deve controlar a execução dos diversos programas e o uso concorrente do processador.

Nos sistemas multiprogramáveis, os processos são executados concorrentemente, compartilhando, entre outros recursos, a utilização do processador, da memória principal e dos dispositivos de E/S. Além disso, em sistemas com múltiplos processadores, não só existe a concorrência de processos pelo uso do processador, como também a execução simultânea de processos nos diferentes processadores.

5.2. Estrutura do processo

Inicialmente, um processo pode ser entendido como um programa em execução; porém, seu conceito é mais abrangente. Em um sistema multiusuário, cada usuário está associado a um processo. Ao executar um

programa, o usuário tem a impressão de possuir o processador e os demais recursos de hardware de forma exclusiva. Na verdade, todos esses recursos estão sendo compartilhados. Nesse caso, o sistema operacional faz com que um processo de usuário seja executado durante certo intervalo de tempo e, após esse intervalo, o processo é interrompido para que outro possa ser executado.

Para que essa troca de processos ocorra sem problemas, é necessário salvar as informações do processo interrompido, para que possam ser recuperadas quando aquele processo voltar a executar. Isso garantirá que o estado do processo seja mantido e que ele possa continuar do ponto onde parou, como se nada tivesse acontecido.

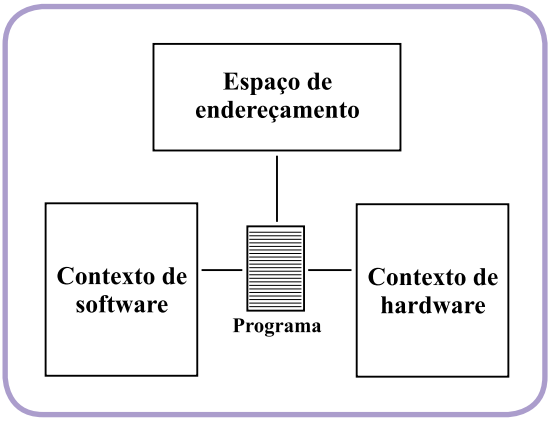

Figura 5-26: Estrutura do processo
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Todas as informações relevantes e necessárias à execução de um programa fazem parte do processo. Essas informações são chamadas de contexto do processo. Além disso, é necessário haver áreas de memória (para armazenamento de instruções e dados) alocadas para o processo. Esse espaço de memória é chamado de espaço de endereçamento.

Conceitos

Processo pode ser definido como um programa em execução, juntamente com seu contexto e suas áreas de dados, código e pilha [4].

O contexto de um processo pode ser dividido em duas partes: o contexto de hardware e o contexto de software. O contexto de hardware armazena o conteúdo de todos os registradores. Já o contexto de software armazena informações sobre a identificação, as quotas e os privilégios do processo.

5.2.1 Contexto de hardware

O contexto de hardware armazena o conteúdo dos registradores gerais da CPU, além dos registradores de uso específico, como *program couter* (PC), *stack pointer* (SP) e registrador de status. Quando um processo está em execução, o conteúdo de alguns registradores é alterado a cada nova instrução executada. No momento em que o processo perde a utilização da CPU, o sistema salva as informações no contexto de hardware do processo [1].

O contexto de hardware é fundamental para a implementação dos sistemas multiprogramáveis, onde os processos se revezam na utilização da CPU, podendo ser interrompidos e, posteriormente, restaurados. A troca de um processo por outro no processador, comandada pelo sistema operacional, é denominada chaveamento (mudança) de contexto.

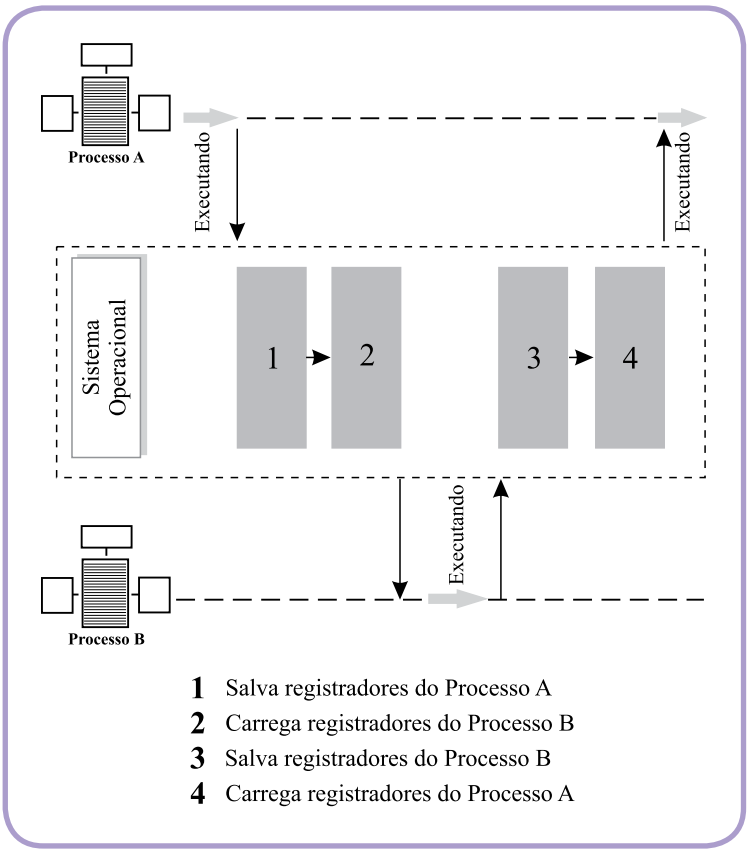

Figura 5-27: Chaveamento de contexto de hardware
Fonte: [1] – Machado e Maia, 2004. Adaptação.

O chaveamento de contexto consiste em salvar o conteúdo dos registradores do processo que está deixando a CPU e carregá-los com os valores referentes ao do novo processo que será executado. Em outras palavras, o chaveamento de contexto resume-se em salvar o contexto de hardware de um processo e restaurar o contexto de hardware de outro.

5.2.2. Contexto de software

O *contexto de software* armazena as informações sobre características e limites dos recursos que podem ser alocados pelo processo, como o número máximo de arquivos abertos simultaneamente e a prioridade de execução e tamanho do buffer para operações de E/S. A maioria dessas características é determinada pelo sistema operacional no momento da criação do processo, enquanto outras podem ser alteradas durante sua existência [1].

A maior parte das informações do contexto de software do processo é proveniente de um arquivo do sistema operacional, conhecido como arquivo de contas. Nesse arquivo, gerenciado pelo administrador de sistema, são especificados os limites dos recursos que cada processo pode alocar. Outras informações presentes no contexto de software são geradas dinamicamente, ao longo da execução do processo.

O contexto de software é composto por três grupos de informações sobre o processo: identificação, quotas e privilégios.

a) Identificação

Cada processo criado pelo sistema, recebe uma identificação única (PID- *Process Identification*), representada por um número. Por meio do PID, o sistema operacional e outros processos podem fazer referência a qualquer processo existente, consultando seu contexto ou alterando uma de suas características. Alguns sistemas, além do PID, identificam o processo através de um nome.

O processo também possui a identificação do usuário ou do processo que o criou (*owner*). Cada usuário possui uma identificação única no sistema (UID – *User Identification*), atribuída ao processo no momento de sua criação. Pela UID, é possível implementar um modelo de segurança em que apenas os objetos (processos, arquivos, áreas de memória, etc.) que possuam a mesma UID possam ser acessados, impedindo que um usuário acesse informações sobre objetos de outro usuário.

b) Quotas

As quotas são os limites de cada recurso do sistema que um processo pode alocar. Alguns exemplos de quotas presentes na maioria dos sistemas operacionais são:

- Tamanho máximo de memória principal e secundária que o processo pode alocar;

- Número máximo de arquivos abertos simultaneamente;
- Número máximo de operações de E/S pendentes;
- Tamanho máximo de processo, subprocessos e *threads* que podem ser criados.

Caso uma quota seja insuficiente, o processo poderá ser executado lentamente, interrompido durante seu processamento, ou mesmo, não ser executado.

c) Privilégios

Os privilégios ou direitos definem as ações que um processo pode fazer em relação a ele mesmo, aos demais processos e ao sistema operacional.

Privilégios que afetam o próprio processo permitem que suas características possam ser alteradas, como prioridade de execução, limites alocados na memória principal e secundária, etc. Já os privilégios que afetam os demais processos permitem, além da alteração de suas próprias características, a alteração de outros processos.

Privilégios que afetam o sistema são os mais amplos e poderosos, pois estão relacionadas à operação e gerência do ambiente, como a desativação do sistema, alteração de regras de segurança, criação de outros processos privilegiados, modificação de parâmetros de configuração do sistema, entre outros. A maioria dos sistemas operacionais possui uma conta de acesso com todos esses privilégios disponíveis, com o propósito de o administrador gerenciar o sistema operacional. No sistema Unix existe a conta "root"; no Windows 2000/XP, a conta "administrador"; e no OpenVMS, a conta "*system*" com esse perfil.

5.2.3 Espaço de endereçamento

O *espaço de endereçamento* é a área de memória pertencente ao processo onde às instruções e os dados do programa são armazenados para execução. Cada processo possui seu próprio espaço de endereçamento, que deve ser devidamente protegido do acesso dos demais processos. Nos capítulos seguintes serão analisados diversos mecanismos de implementação e administração do espaço de endereçamento.

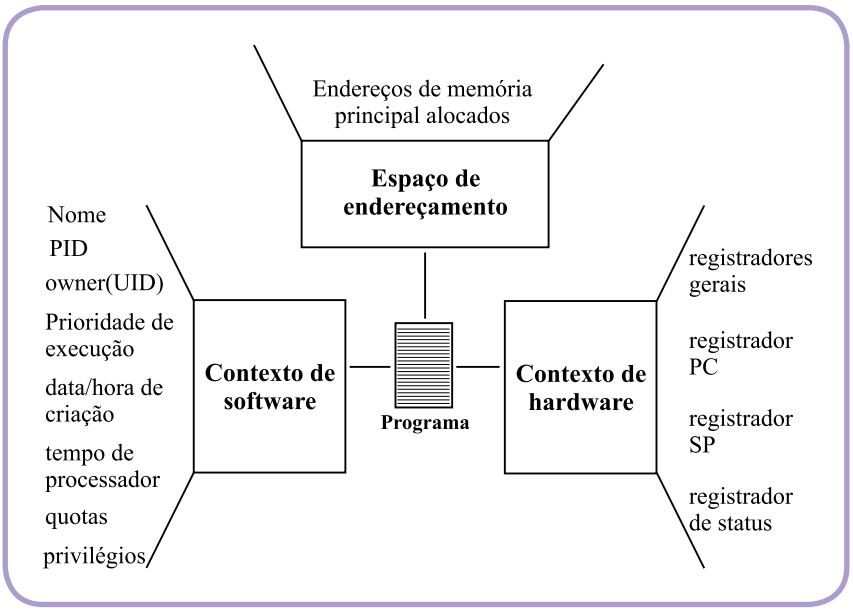

Figura 5-28: Estrutura do processo e suas informações
Fonte: [1] – Machado e Maia, 2004. Adaptação.

5.3. Bloco de controle do processo

O sistema operacional utiliza uma estrutura de dados, chamada *bloco de controle de processo (Process Control Block – PCB)*, para implementar os processos. A partir do PCB, o sistema operacional mantém todas as informações sobre o contexto de hardware, o contexto de software e o espaço de endereçamento de cada processo.

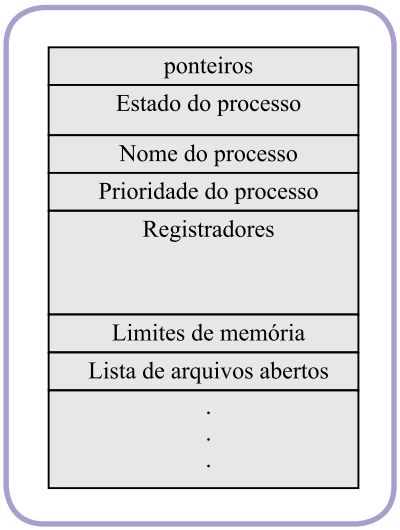

Figura 5-29: Bloco de controle do processo (PCB)
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Os PCB's de todos os processos residem na memória principal, em uma área exclusiva do sistema operacional. O tamanho dessa área geralmente é limitado por um parâmetro do sistema operacional que permite

especificar o número máximo de processos possíveis de serem suportados simultaneamente pelo sistema. Toda a gerência dos processos é realizada por meio de chamadas de sistema, que realizam operações como criação, alteração de características, visualização, eliminação, sincronização e suspensão de processos, entre outras.

5.4. Estados do processo

Em sistemas multiprogramáveis, é comum haver vários processos compartilhando a utilização do processador. Para evitar que algum processo monopolize o processador, o sistema operacional determina quando cada processo tem direito a executar. Assim, enquanto um processo executa, os demais aguardam em uma fila pela sua vez. Se um processo estiver executando e solicitar uma operação de E/S (através uma chamada de sistema), ele deverá liberar o processador e aguardar até que a operação seja concluída. Só então ele estará apto novamente a disputar a utilização do processador com os outros processos.

Para saber em que situação se encontra cada processo, o sistema operacional implementa o conceito de estados do processo. Ao longo do seu processamento, os processos passam por diferentes estados, em função de eventos gerados pelo próprio processo ou pelo sistema operacional.

5.4.1. Estados de um processo ativo

Um processo ativo pode se encontrar em três diferentes estados: execução (*running*), pronto (*ready*) e espera (*wait*).

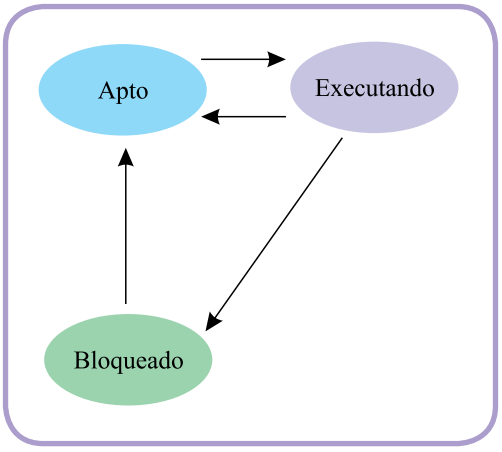

Figura 5-30: Estados de um processo ativo
Fonte: [4] – Oliveira, Carissimi e Toscani, 2004. Adaptação

a) Executando (*running*)

Considera-se que um processo está no *estado de execução*, quando ele está sendo processado pela CPU. Em sistemas com um único processador, somente um processo pode ser executado em um dado momento. O sistema operacional alterna (escalona) a utilização do processador entre os processos, segundo alguma política estabelecida por ele.

Em sistemas com múltiplos processadores, existe a possibilidade de mais de um processo estar sendo executado ao mesmo tempo. Nesse tipo de sistema, também é possível um mesmo processo ser executado simultaneamente, em mais de uma CPU (processamento paralelo).

b) Apto ou Pronto (*ready*)

Um processo está no *estado de apto* ou *pronto*, quando aguarda sua vez de ser executado. Em geral, existem vários processos no sistema em estado de pronto, organizados em uma fila. Por meio dessa fila, chamada de *fila de aptos* ou *fila de prontos*, o sistema operacional determina a ordem e os critérios pelos quais os processos em estado de pronto devem utilizar o processador. Esse mecanismo é conhecido como **escalonamento**. Existem vários critérios de escalonamento de processos, tais como: fila simples (FIFO), circular, fila com prioridades, etc. Tais critérios serão vistos posteriormente. Neste material, os termos fila de prontos e fila de aptos serão utilizados como sinônimos.

c) Bloqueado ou Espera (*wait*)

Um processo no *estado bloqueado* ou *espera* aguarda por algum evento externo ou por algum recurso para prosseguir seu processamento. Como exemplo, podemos citar um processo que aguarda o término de uma operação de entrada/saída ou a espera de uma determinada data e/ou hora para continuar sua execução.

O sistema operacional organiza os vários processos no estado de espera em listas. Quando ocorre um evento, o sistema operacional verifica qual processo estava aguardando aquele evento, e o processo é transferido para o estado de pronto. Em outras palavras, o processo volta à fila de aptos e está pronto para prosseguir sua execução.

5.4.2. Estados de criação e destruição de um processo

Para um processo entrar na fila de aptos e aguardar sua vez de executar, é necessário que seja criado pelo (*loader*) sistema operacional. Primeiro o sistema operacional deve alocar na memória principal as áreas de

código, dados e pilha. Em seguida, o programa deve ser transferido da memória secundária para a memória principal nas áreas alocadas. Por último, o PCB é criado e inicializado apropriadamente, com as informações do processo. A partir desse momento, o sistema operacional já reconhece a existência do processo.

Após criar o processo, o sistema operacional normalmente o coloca na fila de aptos para que ele concorra à utilização do processador. No entanto, algumas vezes, podem faltar recursos (por exemplo: número máximo de processos ativos alcançados); nesse caso, o sistema operacional manterá o processo no estado de criação, até que haja recursos para ele se tornar ativo.

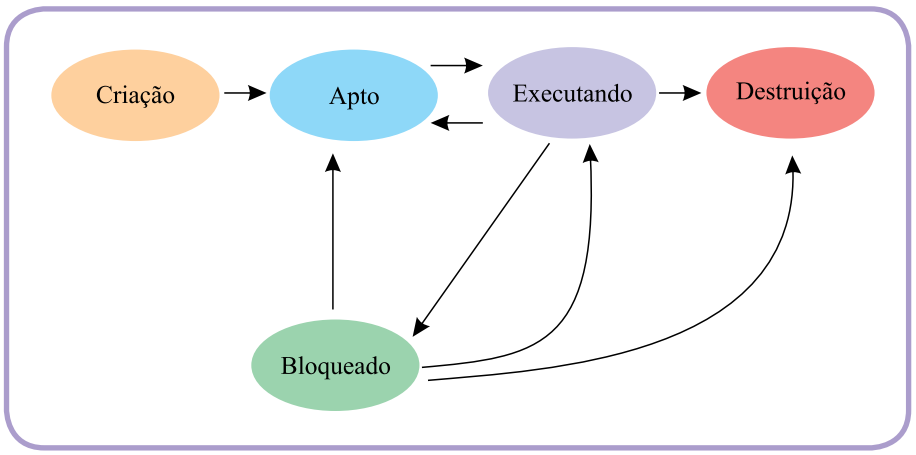

Figura 5-31: Estados do processo
Fonte: [4] – Oliveira, Carissimi e Toscani, 2004. Adaptação

Quando um processo é finalizado, todos os recursos do processo devem ser desalocados e o PCB deve ser eliminado pelo sistema operacional. O término de um processo pode ocorrer pelas seguintes razões:

- Término normal de execução;
- Eliminação forçada por erros de proteção;
- Eliminação por outro processo;
- Eliminação forçada por ausência de recursos disponíveis no sistema.

Algumas vezes, o sistema operacional pode manter em memória as informações de um processo finalizado. Nesse caso, o processo não é considerado mais ativo, mas, como o PCB ainda existe, o sistema operacional pode obter informações sobre o processo, como contabilização e uso de recursos. Após as informações serem extraídas, o processo, geralmente, deixa de existir.

5.5. Transições de estado do processo

Um processo muda de estado durante seu processamento, em função de eventos originados por ele próprio (*eventos voluntários)* ou pelo sistema operacional *(eventos involuntários)*. Basicamente, existem quatro mudanças de estado que podem ocorrer a um processo.

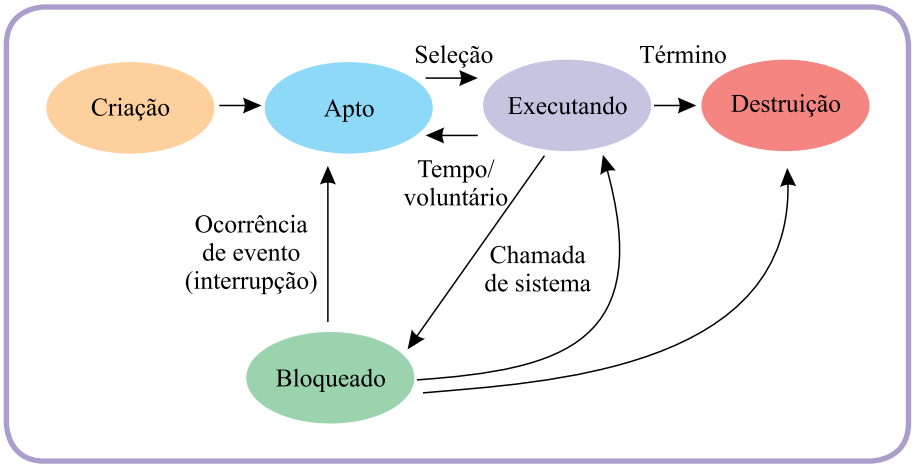

Figura 5-32: Transição de Estados do processo
Fonte: [4] – Oliveira, Carissimi e Toscani, 2004. Adaptação

5.5.1. Apto → executando

Após a criação de um processo, o sistema o coloca em uma fila de processos no estado de apto, onde aguarda por uma oportunidade para ser executado. Cada sistema operacional tem seus próprios critérios e algoritmos para a escolha da ordem em que os processos serão executados (política de escalonamento). Esses critérios e seus algoritmos serão analisados em detalhes nos capítulos seguintes.

A transferência de um processo do estado apto para o estado de execução indica que outro processo saiu do estado de execução (perdeu o processador). Um processo perde o processador quando sua fatia de tempo (*time slice*) termina, quando faz uma chamada de sistema ou quando chega ao fim de sua execução (normal ou forçada). Isso faz com que o sistema operacional selecione um processo no estado apto para ser executado.

5.5.2. Execução → apto

Um processo em execução passa para o estado de apto por eventos gerados pelo sistema, como o término da sua fatia de tempo (*time slice*) ou para ceder a vez a um processo de maior prioridade. Nesse caso, o processo volta para a fila de aptos, onde aguarda por uma nova oportunidade para continuar seu processamento.

5.5.3. Execução → bloqueado

Um processo em execução pode passar para o estado de bloqueado, por eventos gerados pelo próprio processo. Por exemplo, quando um processo em execução necessita realizar uma operação de E/S, realiza a solicitação por meio de uma chamada de sistema. Ao fazer isso, o sistema operacional é chamado, retira o processo da execução e o coloca no estado bloqueado. Em seguida, instrui o controlador de dispositivos de E/S, para que a operação seja realizada, e seleciona um processo da fila de aptos, para executar. O processo que foi bloqueado aguardará nesse estado até que sua operação de E/S seja concluída.

Além disso, um processo em execução também pode passar para o estado de bloqueado em função de eventos externos. Um evento externo é gerado, por exemplo, quando o sistema operacional suspende, por um período de tempo, a execução de um processo.

5.5.4. Bloqueado → apto

Um processo no estado bloqueado passa para o estado de apto quando a operação solicitada é atendida ou o recurso esperado é concedido. Por exemplo, um processo que solicitou uma operação de E/S ficará no estado bloqueado até que o controlador de dispositivos informe que a operação foi concluída. O controlador sinaliza a conclusão da operação de E/S através de uma interrupção de hardware que, imediatamente, chama o sistema operacional para transferir o respectivo processo do estado bloqueado para apto.

Em geral, um processo no estado bloqueado sempre terá de passar pelo estado de apto antes de poder ser novamente selecionado para execução. Normalmente não existe a mudança do estado bloqueado para o estado de execução diretamente. Em casos raros, um processo poderá fazer isso, caso a chamada de sistema seja extremamente rápida como a leitura da hora do sistema.

5.5.5. Transições de estado com swapping

Um processo no estado apto ou bloqueado pode não se encontrar residente na memória principal. Essa condição ocorre quando não existe espaço suficiente para todos os processos na memória principal e parte do contexto do processo é levada para a memória secundária. Uma técnica conhecida como ***swapping*** retira o processo da memória principal e o traz de volta, seguindo os critérios de cada sistema operacional. Nesse caso, os processos em estado bloqueado e apto podem estar **residentes** ou **não-residentes** na memória principal.

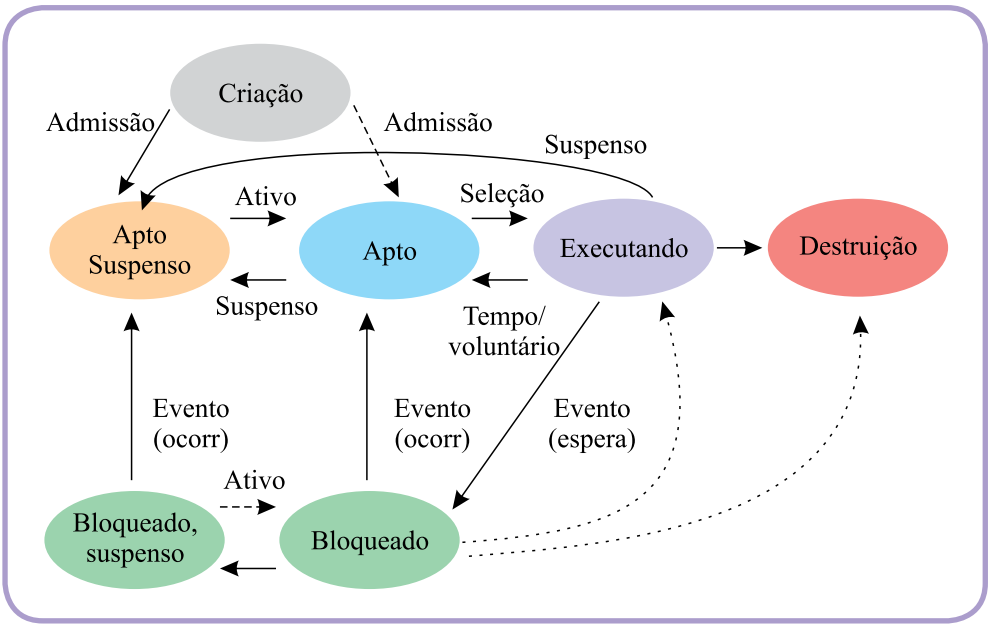

Figura 5-33: Transições de estado com Swapping
Fonte: [4] – Oliveira, Carissimi e Toscani, 2004. Adaptação

Atividades

Atividades

1. Defina o conceito de processo.
2. Por que o conceito de processo é tão importante no projeto de sistemas multiprogramáveis?
3. É possível que um programa execute no contexto de um processo e não execute no contexto de um outro? Por quê?
4. Quais partes compõem um processo?
5. O que é o contexto de hardware de um processo e como é a implementação da troca de contexto?
6. Qual a função do contexto de software? Exemplifique cada grupo de informação.
7. O que é o espaço de endereçamento de um processo?
8. Como o sistema operacional implementa o conceito de processo? Qual a estrutura de dados indicada para organizar os diversos processos na memória principal?
9. Defina os cinco estados possíveis de um processo.
10. Dê um exemplo que apresente todas as mudanças de estado de um processo, juntamente com o evento associado a cada mudança.

5.6 Processos independentes, subprocessos e threads

Processos independentes, subprocessos e *threads* são maneiras diferentes de implementar a concorrência dentro de uma aplicação. Nesse caso, busca-se subdividir o código em partes para trabalharem de forma cooperativa. Em aplicações acessadas simultaneamente por vários usuários, a concorrência pode proporcionar um tempo de espera menor, melhorando o desempenho da aplicação e beneficiando os usuários.

O uso de **processos independentes** é a maneira mais simples de implementar a concorrência em sistemas multiprogramáveis. Nesse caso não existe vínculo entre o processo criado e seu criador. A criação de um processo independente exige a alocação de um PCB, possuindo contextos de hardware, contexto de software e espaço de endereçamento próprio [1].

Subprocessos são processos criados dentro de uma estrutura hierárquica. Nesse modo, o processo criador é denominado **processo pai** e o novo processo é chamado de **processo filho** ou **subprocesso**. O subprocesso pode criar outras estruturas de subprocessos. Uma característica dessa implementação é a dependência existente entre o processo criador e o subprocesso. Caso um processo pai deixe de existir, o sistema operacional pode ser configurado para eliminar automaticamente os processos filhos. Semelhante aos processos independentes, subprocessos possuem seu próprio PCB.

Além da dependência hierárquica entre processos e subprocessos, uma outra característica nesse tipo de implementação é que subprocessos podem compartilhar quotas com o processo pai. Nesse caso, quando um subprocesso é criado, o processo pai cede parte de suas quotas ao processo filho.

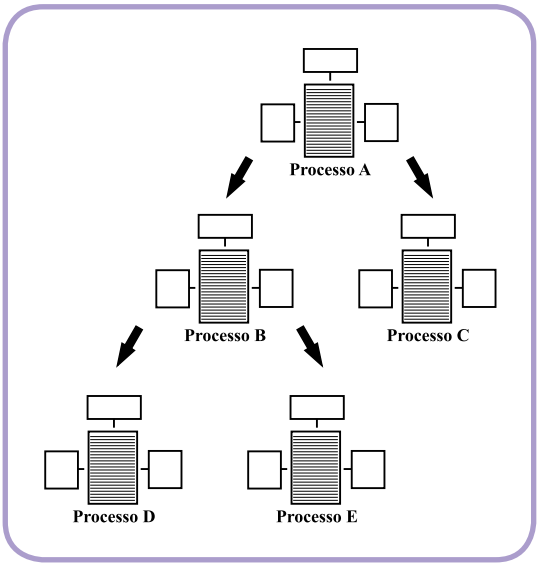

Figura 5-34: Processos dependentes
Fonte: [1] – Machado e Maia, 2004. Adaptação.

O uso de processos independentes e subprocessos, no desenvolvimento de aplicações concorrentes, demandam consumo de diversos recursos do sistema. Sempre que um novo processo é criado, o sistema deve alocar recursos (contexto de hardware, contexto de software e espaço de endereçamento), consumindo tempo de CPU nesse trabalho. No momento do término dos processos, o sistema operacional também dispensa tempo para desalocar recursos previamente alocados. Outro problema existente é a comunicação e a sincronização entre processos, considerada pouco eficiente, visto que cada processo possui seu próprio espaço de endereçamento.

O *thread* representa uma tentativa de reduzir o tempo gasto na criação, eliminação e chaveamento de contexto de processos nas aplicações concorrentes, bem como de economizar recursos do sistema. Em um ambiente *multithread*, um único processo pode suportar múltiplos *threads*, cada qual associado a uma parte do código da aplicação. Nesse caso, não é necessário haver diversos processos para a implementação da concorrência. *Threads* compartilham o processador da mesma maneira que um processo, ou seja, enquanto um *thread* espera por uma operação de E/S, outro *thread* pode ser executado.

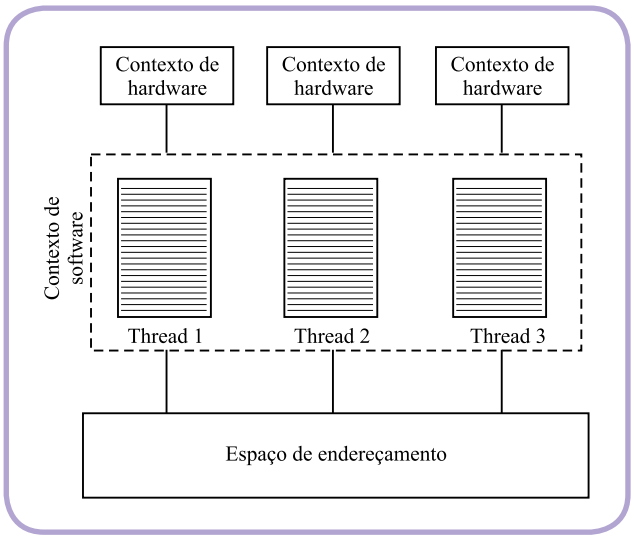

Figura 5-35: Processo multithread
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Cada *thread* possui seu próprio contexto de hardware, porém, compartilha o mesmo contexto de software e espaço de endereçamento como os demais *threads* do processo. Por compartilharem o espaço de endereçamento, a comunicação entre os *threads* de um mesmo processo pode ser feita de forma simples e rápida. E o compartilhamento do contexto de software torna mais rápido a criação, a eliminação e o chaveamento de contexto dos *threads*.

5.7. Processos *foreground* e *background* e pipes

Um processo possui, sempre associado à sua estrutura, pelo menos dois canais de comunicação por onde são realizadas todas as entradas e saídas de dados ao longo do seu processamento. Os canais de entrada (*input*) e saída (*output*) de dados podem estar associados a terminais, arquivos, impressoras e até mesmo a outros processos.

5.7.1. Processo *foreground*

Um *processo foreground* é aquele que permite a comunicação direta do usuário com o processo durante o seu processamento. Nesse caso, tanto o canal de entrada quanto o de saída estão associados a um terminal com teclado, mouse e monitor, permitindo, assim, a interação com o usuário. O processamento interativo tem, como base, processos *foreground*.

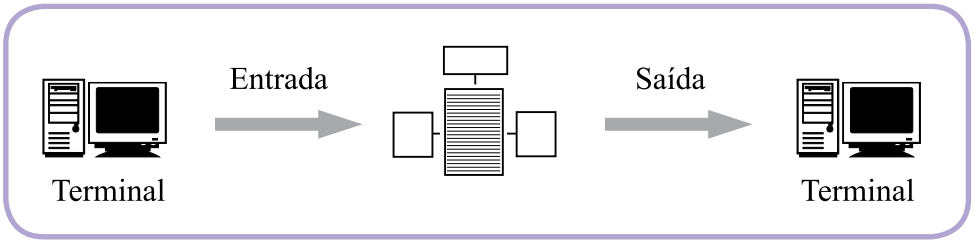

Figura 5-36: Processo ***foreground***
Fonte: [1] – Machado e Maia, 2004. Adaptação.

5.7.2. Processo *background*

Um *processo background* é aquele em que não existe a comunicação com o usuário durante o seu processamento. Nesse caso, os canais de E/S não estão associados a nenhum dispositivo de E/S interativo, mas, em geral, a arquivos de E/S. Em outras palavras, não há interação com o usuário, pois tanto a entrada quanto a saída são realizadas por meio de arquivos. O processamento do tipo batch é realizado através de processos *background*.

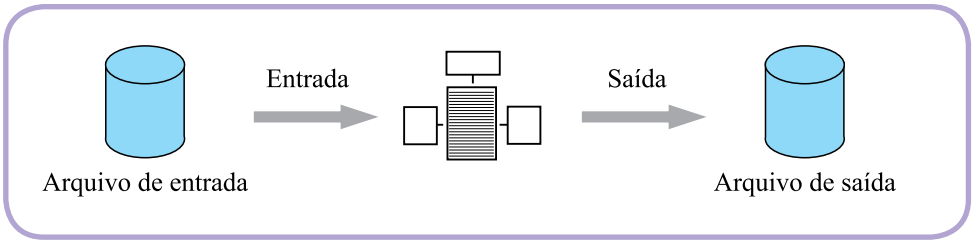

Figura 5-37: Processo ***background***
Fonte: [1] – Machado e Maia, 2004. Adaptação.

5.7.3. *Pipe* entre processos

É possível associar o canal de saída de um processo ao canal de entrada de um outro processo. Nesse caso, dizemos que existe um *pipe* ligando os dois processos. Por exemplo, se um processo A gera uma listagem e um processo B tem como função ordená-la, basta associar o canal de saída do processo A ao canal de entrada do processo B.

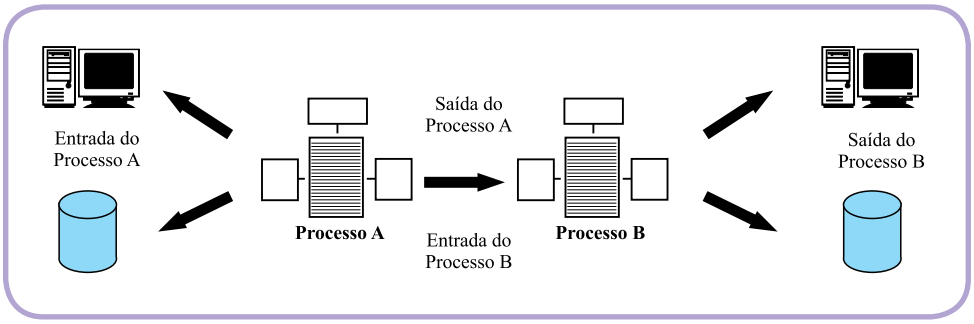

Figura 5-38: ***Pipe*** entre processos
Fonte: [1] – Machado e Maia, 2004. Adaptação.

5.8 Processos do sistema operacional

O conceito de processo, além de estar associado às aplicações de usuários, também pode ser implementado na própria arquitetura do sistema operacional. Como visto anteriormente, a arquitetura microkernel implementa o uso intensivo de processos que disponibilizam serviços para outros processos (das aplicações e do próprio sistema operacional).

Quando processos são utilizados para a implementação de serviços do sistema, estamos retirando código do seu núcleo, tornando-o menor e mais estável. No caso de um ou mais serviços não serem desejados, basta não ativar os processos responsáveis, o que permitirá liberar memória para os processos dos usuários.

Segue uma listagem de alguns serviços que o sistema operacional pode implementar por meio de processos:

- Auditoria e segurança;
- Serviços de rede;
- Contabilização do uso de recursos;
- Contabilização de erros;
- Gerência de impressão;

- Gerência de *jobs batch*;
- Temporização;
- Comunicação de eventos;
- Interface de comandos (shell).

5.9. Processos CPU-bound e I/O-bound

Os processos podem ser classificados como CPU-bound ou I/O-bound, de acordo com a utilização do processador e dos dispositivos de E/S [1, 2, 3, 4].

Um processo é definido como CPU-bound (ligado à CPU) quando passa a maior parte do tempo no estado de execução, ou seja, utilizando o processador. Esse tipo de processo realiza poucas operações de leitura e gravação e é encontrado em aplicações científicas que efetuam muitos cálculos.

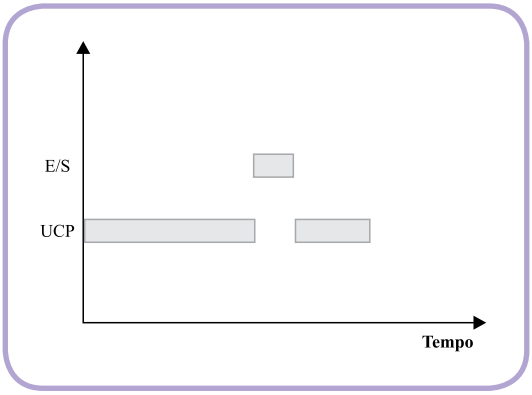

Figura 5-39: Processo CPU-bound
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Um processo é classificado como I/O-bound (ligado a E/S) quando passa a maior parte do tempo no estado bloqueado, pois realiza um elevado número de operações de E/S. Esse tipo de processo é encontrado em aplicações comerciais, que se baseiam em leitura, processamento e gravação.

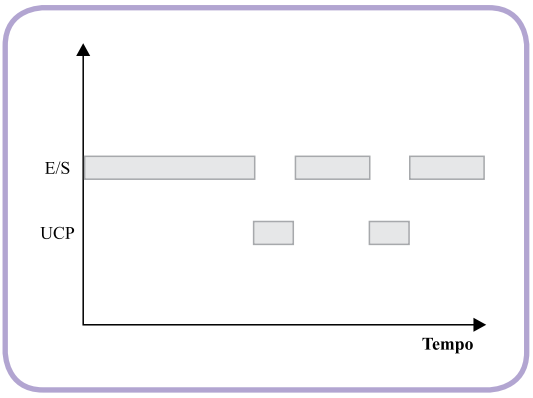

Figura 5-40: Processo I/O-bound
Fonte: [1] – Machado e Maia, 2004. Adaptação.

Os processos interativos também são bons exemplos de processos I/O-bound, pela forma de comunicação entre o usuário e o sistema, normalmente lenta, devido à utilização de terminais.

5.10. Sinais

O uso de *Sinais* é um mecanismo que permite notificar processos sobre eventos gerados pelo sistema operacional ou por outros processos. A utilização de sinais é fundamental para a gerência de processos, além de possibilitar a comunicação e sincronização entre processos [1].

Fala Professor

Um exemplo de uso de sinais é quando alguém utiliza uma sequência de caracteres do teclado, como [Ctrl-C], para interromper a execução de um programa. Nesse caso, o usuário gerou uma interrupção de hardware (teclado) por meio da digitação dessa combinação de teclas e, a partir dela, o sistema operacional gera um sinal indicando ao processo a ocorrência do evento. No momento em que o processo identifica a chegada do sinal, uma rotina específica de tratamento é executada.

A maior parte dos eventos associados a sinais é gerada pelo sistema operacional ou pelo hardware, como a ocorrência de exceções, interrupções geradas por terminais, limites de quotas excedidos durante a execução dos processos e alarmes de tempo. Em outras situações, os eventos são gerados a partir de outros processos, com o propósito de sincronizar suas execuções.

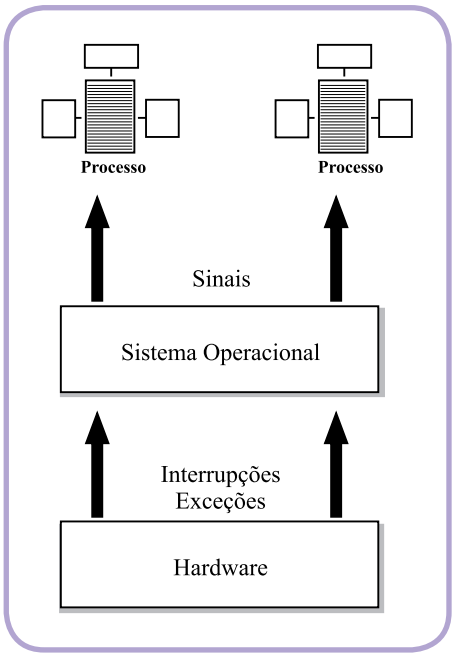

Figura 5-41: Sinais
Fonte: [1] – Machado e Maia, 2004. Adaptação.

A geração de um sinal ocorre quando o sistema operacional, a partir da ocorrência de eventos síncronos ou assíncronos, notifica o processo através de bits de sinalização localizados no seu PCB. Um processo não responde instantaneamente a um sinal. Os sinais ficam pendentes até que o processo seja escalonado, quando então serão tratados. Por exemplo, quando um processo é eliminado, o sistema ativa o bit, associado a esse evento. O processo somente será excluído do sistema quando for selecionado para execução. Nesse caso, é possível que o processo demore algum período de tempo até ser eliminado de fato.

O tratamento de um sinal é muito semelhante ao mecanismo de interrupções. Quando um sinal é tratado, o contexto do processo é salvo e a execução desviada para um código de tratamento de sinal (*signal handler*), geralmente no núcleo do sistema. Após a execução do tratador de sinais, o programa pode voltar a ser processado do ponto onde foi interrompido.

Em certas implementações, o próprio processo pode tratar o sinal por meio de rotinas de tratamento definidas no código do próprio programa. É possível também que um processo bloqueie temporariamente ou ignore por completo alguns sinais. O mecanismo de sinais assemelha-se ao tratamento de interrupções e exceções, vistos anteriormente, porém, com propósitos diferentes. O sinal está para o processo, assim como as interrupções e exceções estão pra o sistema operacional.

Indicações

[1] MACHADO, F.B. e MAIA, L.P. ***Arquitetura de Sistemas Operacionais***. 4.ed. LTC, 2007.

[2] SILBERSCHATZ, A., GALVIN, P.B., GAGNE, G. ***Fundamentos de Sistemas Operacionais***. 6.ed. LTC, 2004.

[3] TANENBAUM, A.S. ***Sistemas Operacionais Modernos***. 2.ed. Pearson Brasil, 2007.

[4] OLIVEIRA, R.S., CARISSIMI, A.S., TOSCANI, S.S. ***Sistemas Operacionais***. 3.ed. Sagra-Luzzato. 2004.

Atividades

Atividades

1. Diferencie processos multithreads, subprocessos e processos independentes.
2. Explique a diferença entre processos *foreground* e *background*.
3. Qual a relação entre processo e a arquitetura microkernel?
4. Dê exemplos de aplicações CPU-bound e I/O-bound.
5. Justifique com um exemplo a frase "o sinal está para o processo, assim como as interrupções e exceções estão para o sistema operacional".
6. Explique como a eliminação de um processo utiliza o mecanismo de sinais.